VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Kontrola kvality výrobků pomocí počítačového vidění

Goods Quality Evaluation Using Computer Vision

2017 Bc. Petr Kresta

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Kresta**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Kontrola kvality výrobků pomocí počítačového vidění
Goods Quality Evaluation Using Computer Vision

Jazyk vypracování: čeština

Zásady pro vypracování:

Zajištění kvality průmyslových výrobků je často realizováno pomocí počítačového vidění, kdy se měří různé vlastnosti z obrazů výrobků a kontroluje se, zdali jsou v zadaném limitu. Cílem práce je implementace aplikace pro vyhodnocování kvality svarů kovových plátů z fotografií jejich příčného řezu.

Ve své práci proveďte:
1. Nastudujte problém vyhodnocování kvality svarů železných plátů (výrobek).
2. Seznamte se s postupy pro vyhledávání zájmových bodů nebo čar [1, 2, 3] v obraze s výrobkem.
3. Implementujte algoritmy pro měření parametrů výrobků jako jsou např.: převýšení svaru, přesazení železných plátů, přetečení svaru, apod. z norem EN ISO 5817 a EN ISO 3183.
4. Svou implementaci řádně otestujte na množině obrazových dat poskytnutých vedoucím DP. Své závěry náležitě popište v textu práce.

Seznam doporučené odborné literatury:

[1] Nieto, M., Cuevas, C., Salgado, L., Garcia, N.: Line Segment Detection Using Weighted Mean Shift Procedures on a 2D Slice Sampling Strategy, Pattern Analysis and Applications, 14 (2), pp. 149-163, 2011
[2] Akinlar, C., Topal, C.: EDLines: A Real-time Line Segment Detector with a False Detection Control, Pattern Recognition Letters, 32 (13), pp. 1633-1642, 2011
[3] von Gioi, R. G., Jakubowicz, J., Morel, J. M., Randall, G.: LSD: A Fast Line Segment Detector with a False Detection Control, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32 (4), pp. 722-732, 2010

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této práce je implementovat aplikaci, kterou bude možno vyhodnocovat parametry dvou svařených železných plátů z norem EN ISO 5817 a EN ISO 3183. Tato aplikace má dle fotografie příčného řezu dvou svařených železných plátů měřit parametry výrobku. Aplikace bude uživatelsky přívětivá a srozumitelná, čímž povede k usnadnění kontroly kvality svařených plátů a snížení nákladů na tento průmyslový proces.

Klíčová slova: svařování, svar, železné pláty, počítačové vidění, OpenCV, zpracování obrazu, diplomová práce

Abstract

The purpose of this thesis is to implement application which will be measuring weld parameters from norms EN ISO 5817 a EN ISO 3183. As input of application will be used side cut of two welded metal sheets. Implemented software will be intuitive and easy to use, to lower costs of quality check in industrial process.

Key Words: welding, weld, metal sheets, computer vision, OpenCV, image processing

Obsah

Seznam použitých zkratek a symbolů

GUI	–	Graphical User Interface
LSD	–	Line Segment Detector

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V pracovním procesu výroby produktů, které jsou spojovány svařováním a tyto svary jsou namáhány ať v tahu, tlaku, krutu či ohybu, je velice důležité kontrolovat kvalitu svarů. Proto se na každém svaru kontroluje jeho kvalita. Základní proporce, které musí splňovat každý svar, jsou sepsány do norem. Zákazník vytváří produkty, které musí splňovat normy EN ISO 5817 a EN ISO 3183. Z nichž plyne několik parametrů, které musí být splněny, aby mohl být produkt prodán či použit jako část většího produktu. Výrobek, který nesplní normy, nemůže být dále využíván a musí být buď opraven, zdali to umožňuje povaha výrobku nebo musí být roztaven a vyroben znovu. Tento proces bývá na provozovnách měřen klasickými způsoby, které by měla tato aplikace nahradit. Zaměstnanec s použitím posuvného měřítka a úhloměru posuzuje parametry a splnění norem. Tento často opakující proces zabírá zaměstnancům spoustu času, který by mohli využít například pro vylepšení výrobního procesu. Tato aplikace by měla zaměstnancům automatizovat tento proces a tím firmě ušetřit náklady spojené s kontrolou. Díky postupně zlepšující se odnoži informačních technologií, zpracování obrazu, se dá část této kontroly jakosti přesunout na počítače. Nicméně tato automatizace nemůže, být úplně dokonalá, takže jistá asistence zaměstnance bude i nadále zapotřebí. Ovšem celý proces se značně urychlí. V první části práce se budu zabývat metodikami zpracování obrazu, jako je Gaussovo rozmazání, bilaterální filtr, Cannyho hranový detektor. V druhé části se budu zabývat implementací výsledné měření parametrů dle norem a použitím aplikace. Na závěr bude porovnání přesnosti měření klasickým způsobem a touto aplikací.

2 Předzpracování obrazu

Základním problémem ve zpracování obrazu je nekvalitní zdroj. Vstupní obraz může být špatně nasvícen, takže v něm vznikají tmavé až jednolité plochy a objekty, které díky výslednému malému kontrastu od sebe ani nejdou odlišit. Obraz může být i různě rozmazaný ať vlivem pohybu či špatnému zaostření čočky. Díky nekvalitním snímačům v obraze taktéž vzniká šum. Toto všechno zapříčiňuje špatné hledání zájmových bodů a jejich vyhodnocení. Tudíž klíčovou úlohou je vstupní obraz zpracovat tak, aby v podstatě obsahoval jasně a zřetelně jenom potřebné náležitosti. Přestože člověk v takto nedokonalém obrazu dokáže vidět objekty, například na základě předešlých zkušeností, tak počítač žádné předešlé zkušenosti nemá a nemůže si „domýšlet“. V první části tedy budu pojednávat o tomto problému a následně možnostmi jeho řešení, jež jsou klíčové pro výslednou úspěšnost.

2.1 Jasový histogram

Jedním z častých problémů je špatné jasové rozložení v obraze, což znamená, že velká část obrazu je ve stejných či podobných odstínech spektra a ve zbytku není skoro žádná část obrazu. Vstupní obraz1 pak působí jednolitě a má malý kontrast. Každý obraz můžeme popsat jasovým histogramem, což je graf sumarizující počet pixelů se stejnou jasovou hodnotou. Provádí se pro každou barevnou složku obrazu zvlášť. Algoritmus je velice jednoduchý, prochází se každý pixel obrazu a dle jeho jasové hodnoty se inkrementuje odpovídající hodnota v jasovém histogramu. Na obrázku3 je vidět jasový histogram vstupního obrazu ještě před vyrovnáním jasového spektra, kde na ose x jsou jednotlivé jasy v rozmezí $[0 - 255]$ a na ose y je četnost výskytu pixelů s touto jasovou hodnotou. Matematicky lze definovat jako

$$p(i) = n_i \tag{1}$$

kde n_i je počet pixelů se stejnou hodnotou jako i.

2.1.1 Vyrovnání jasového histogramu

Vyrovnáním jasového histogramu rozprostřeme hodnoty jasu pixelů po celém spektru. Algoritmus jasové korekce je následující, mějme jasovou hodnotu jednotlivého pixelu obrazu i, kde $i \in [0, 255]$. Na základě předem spočteného histogramu dle popisu v kapitole Jasový histogram2.1 se spočte kumulativní distribuční funkce *cdf* obrazu dle vzorce

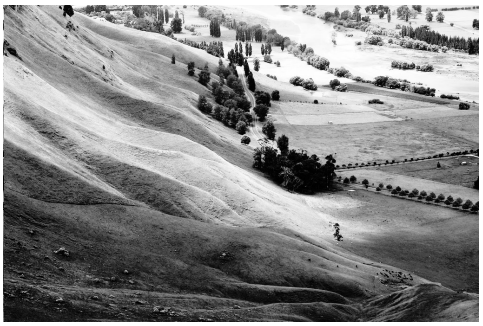
$$cdf(i) = \sum_{(j=0)}^{i} n_j \tag{2}$$

V dalším kroku můžeme spočítat novou hodnotu jasu dle vzorce

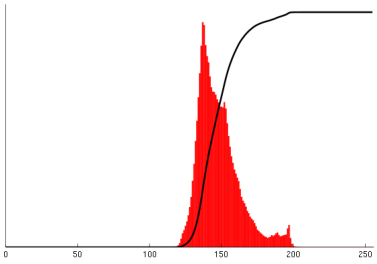
$$h(v) = round(\frac{(cdf(v) - cdf_min)}{((\text{šířka} * \text{výška}) - cdf_min)} * 255) \tag{3}$$

kde v je stará hodnota jasu, cdf_min je nejmenší nenulová hodnota kumulativní distribuční funkce a $h(v)$ je nová hodnota jasu. Tímto postupem se vytvoří nový obraz2 rovnoměrněji rozložený po celém spektru4.

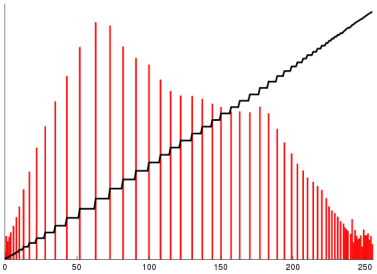
Obrázek 1: Obraz před vyrovnáním histogramu[16]



Obrázek 2: Obraz po vyrovnání histogramu[16]



Obrázek 3: Histogram před vyrovnáním[16]



Obrázek 4: Histogram po vyrovnání[16]

2.2 Redukce šumu

Obraz bývá často zašuměný a tím vnáší problémy například do detekce hran, která může šum považovat za hranu. Takže je důležité se snažit tento šum redukovat co nejvíce. Na redukci šumu existuje spousta metod a některé vybrané budou popsány v následující části.

2.2.1 Konvoluční filtry

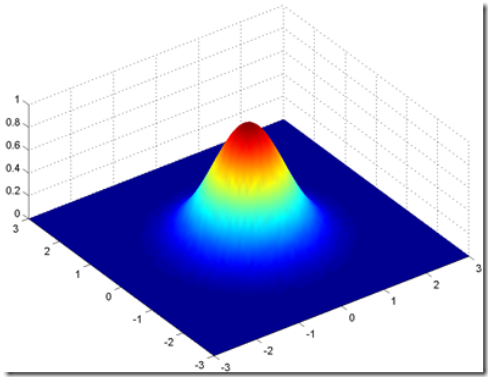
Jak již bylo řečeno, digitální obraz je dvoudimenzionální matice, která v určitém bodě má hodnotu jasu. Proto použijeme dvoudimenzionální vzorec diskrétní konvoluce, který má tvar

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{i=-k}^{k} f(x - i, y - i) * h(i, j) \tag{4}$$

Princip diskrétní konvoluce spočívá v přikládání konvoluční masky postupně na všechny části obrazu a dle jejich hodnot vytvářet obraz nový. Na začátku si definujeme velikost čtvercové masky, třeba na 5x5 body. Maska musí mít lichou velikost, jelikož počítáme vždy hodnotu pro středový pixel. Následně tuto masku naplníme koeficienty, kterými bude původní obraz vynásoben. Nejčastějším rozložením koeficientů je Gaussovo rozložení, které je ve 2D vyobrazeno na na obrazu níže

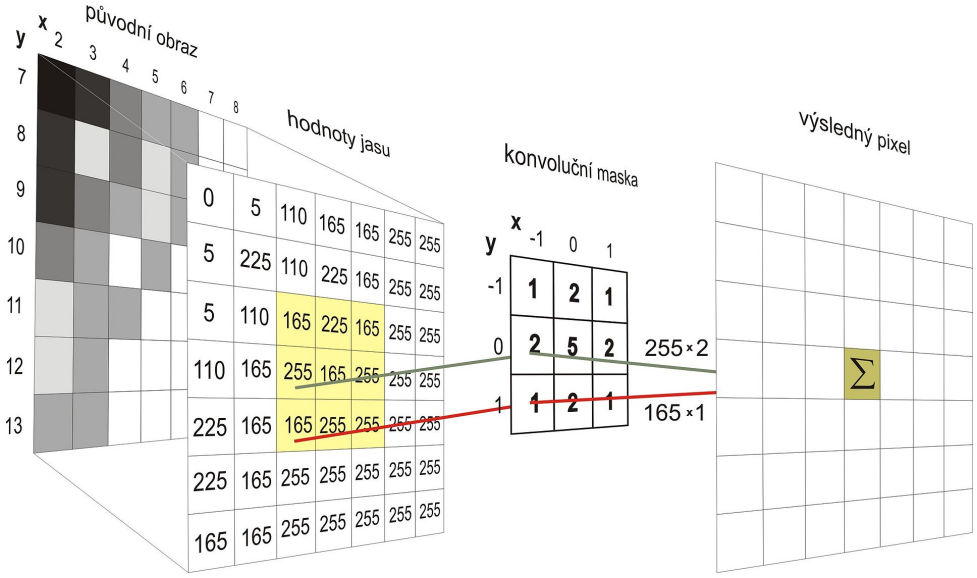
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Obrázek 5: Příklad Gaussovy masky[17]



Obrázek 6: Vizualizace Gaussovy konvoluční masky[18]

Mějme použitou masku z 5 pro následující popis. V každém kroku tuto masku přiložíme na část obrazu, každý pixel masky vynásobíme s pixelem obrazu, následně tyto hodnoty sečteme a případně vynásobíme koeficientem masky, v našem případě vynásobíme číslem $\frac{1}{273}$. Výslednou hodnotu zapíšeme na pozici středového pixelu v nové matici obrazu. Tento postup provádíme, dokud nemáme sestavenou celou výslednou matici.



Obrázek 7: Princip dvou dimenzionální diskrétní konvoluce s maskou 3x3[14]

Výsledkem této konvoluce je rozmazání obrazu, které je závislé na definování a hlavně velikosti masky. Je-li maska větší, tak zabírá do hodnoty pixelu větší sousedství, tudíž více průměruje a tím rozmazává výsledný obraz. Toto je žádaný efekt, díky kterému se můžeme efektivně zbavit šumu, nicméně nevýhodou je právě rozmazávání hran objektu, které jsou pro detekci bodů zájmu důležité.

2.2.2 Bilaterální filtr

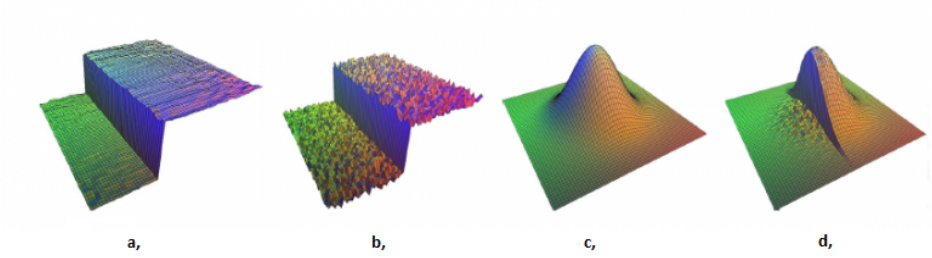
Jako další způsob předzpracování obrazu jsem si vybral bilaterální filtr, který taktéž rozmazává obraz a tím redukuje šum, ovšem na rozdíl od konvoluce Gaussovou maskou tolik nerozmazává hrany objektů. Což je pro detekci hran stěžejní. Základní myšlenkou bilaterálního filtru je, že pro ovlivnění okolních pixelů, nestačí, že je okolí blízké, ale také by měl mít podobnou jasovou hodnotu. Dle knihy[13] je tento filtr popsán vzorcem

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) I_q \tag{5}$$

kde normalizační faktor W_p zajišťuje sumu vah pixelů na 1.0

$$W_p = \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) \tag{6}$$

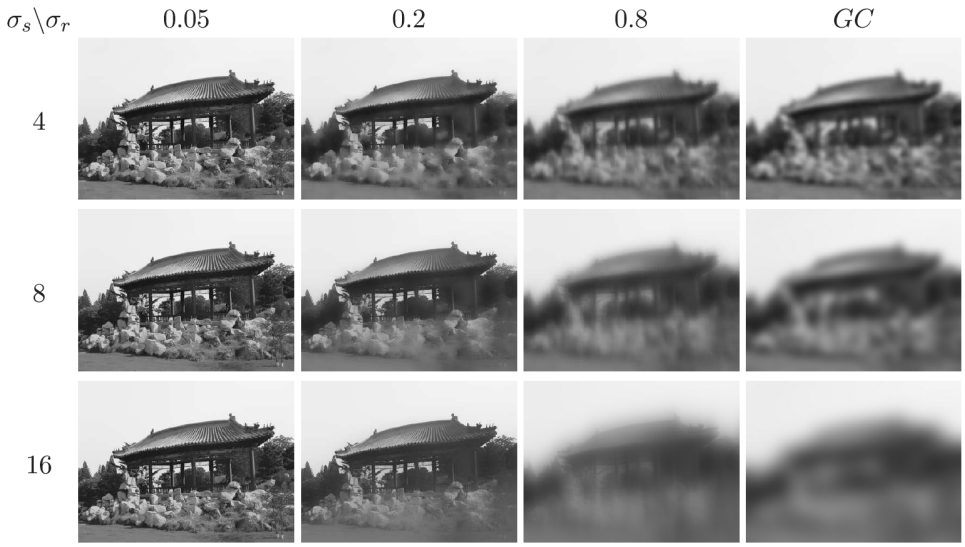
Parametry σ_s a σ_r specifikují úroveň filtrace pro obraz I. Rovnice 5 je normalizovaný vážený průměr, kde G_{σ_s} je prostorové Gaussiánové vážení snižující vliv vzdálených pixelů. G_{σ_r} je rozsah Gaussianu snižující vliv pixelů q pokud se jejich hodnota intenzity liší od I_p.



Obrázek 8: a, $BF[I]_p$ b, I c, G_{σ_s} d, G_{σ_r}[19]

Na obrázku 8 jsou vidět jednotlivé části bilaterálního filtru a jeho výsledek na hraně (obrázek a). U tohoto filtru se nastavují dva parametry: σ_s a σ_r.

- Se zvyšujícím se parametrem σ_r se výsledek bilaterálního filtru blíží k výsledku Gaussově konvoluci
- Zvyšování parametru σ_s má za následek vyhlazování větších ploch



Obrázek 9: Prostorový a vzdálenostní parametr mají za následek větší možnosti rozmazání než Gaussova konvoluce. Pokud nastavím jakýkoli parametr na nulu, tak nedojde k rozmazání. Zvyšování σ_s nebude mít za následek rozmazání hrany, dokud je σ_r menší než amplituda hrany. Na příklad si všimněme, že střecha na obrázku je ostrá pro malé až střední nastavení σ_r a její ostrost je nezávislá na parametru σ_s[20]

(a) Původní obrázek (b) Gaussovo rozostření (c) Bilaterální filtr

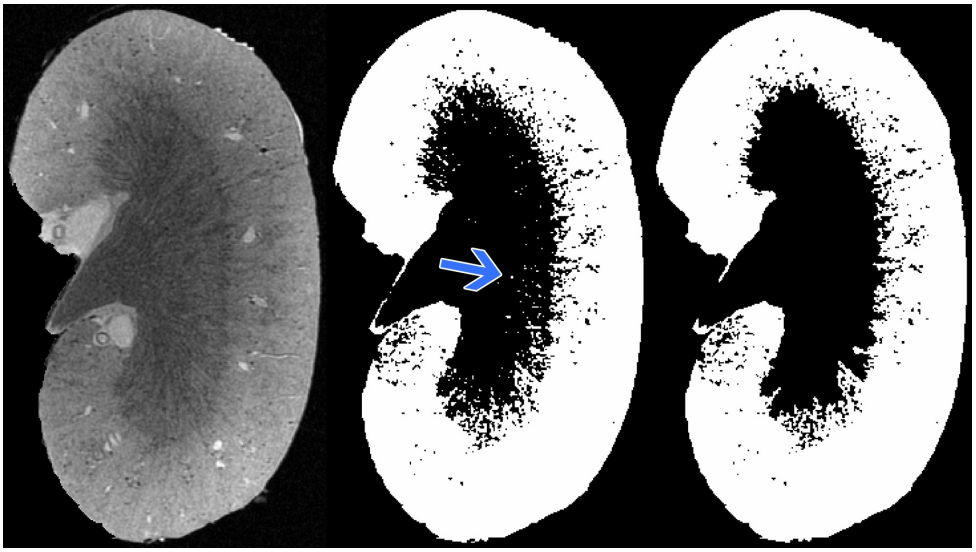
Obrázek 10: Srovnání výsledků Gaussovou konvolucí (b) a při použití bilaterálního filtru (c)[21]

2.3 Prahování

Prahováním se snažíme obraz rozdělit na objekt a pozadí, čímž si, při vhodném nastavení prahu, můžeme hodně usnadnit následující práci s obrazem a stanovování bodů zájmů. Vhodné nastavení prahu je pro správné oddělení objektu od pozadí klíčové. Při nastavení prahu nízko, bude do objektu zahrnuto i pozadí a při příliš vysokém prahu, může docházet k vynechání částí objektu a jejich označení za pozadí. Výsledkem prahování je binární obraz, kde by měl objekt nabývat jedné hodnoty a pozadí hodnoty druhé.

2.3.1 Prahování s hysterézí

Prahování s hysterézí (dvojité prahování) je vylepšená verze standartního prahování, kdy se nastavují dvě hodnoty prahů, vyšší T_H a nižší T_L. Algoritmus pracuje ve dvou krocích, přičemž v první iteraci jsou jsou kandidáti na hranu s hodnotou nad prahem T_H rovnou uznáni za hrany a kandidáti s hodnotou pod prahem T_L rovnou zamítnuti. V druhém kroku se berou v potaz jenom hrany s hodnotou v intervalu $< T_L; T_H >$, u těchto hran se v každé iteraci kontroluje, zdali sousedí s alespoň nějakým bodem již dříve uznaným za hranu. Algoritmus se provádí až do stavu, kdy se výsledný obraz již ve dvou po sobě jdoucích iteracích nezmění. Toto prahování podává značně lepší výsledky než jednoduché prahování jedním prahem, taktéž ho využívá Cannyho hranový detektor2.4.2.



Obrázek 11: Porovnání výsledků jednoduchým prahováním a prahováním s hysterézí[15]

2.3.2 Prahování Otsuovým algoritmem

Mezi globální prahování patři i metoda pana Nobuyuki Otsu. Algoritmus předpokládá, že vstupní obraz v odstínech šedé sestává ze dvou objektů, čímž je popředí a pozadí a hledá nejpřesnější místo v jasovém histogramu, které vhodně rozdělí objekt od pozadí. Výsledkem je binarizovaný obraz (sestávající se z pouze dvou hodnot). Algoritmus je vhodný pro obrazy, které mají dva vrcholy v jasovém histogramu, například objekt je světlý a pozadí tmavé apod, pokud se rozložení jasu v histogramu blíží rovnoměrnému, výsledné rozdělení nebude vhodné. Algoritmus[10] vyhledává práh, který minimalizuje rozdíl v každé třídě a maximalizuje jej mezi nimi.

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{7}$$

Váhy ω_0 a ω_1 jsou pravděpodobnosti, že budou tyto dvě třídy rozděleny prahem t. σ_0^2 a σ_1^2 jsou rozdíly těchto tříd.
Pravděpodobnost třídy $\omega_{0,1}(t)$ je spočtena z L histogramů

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)\omega_1(t) = \sum_{i=t}^{L-1} p(i) \tag{8}$$

Otsu prokázal, že minimalizace rozdílů uvnitř třídy je totéž jako maximalizace rozdílu mezi třídami[11]

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{9}$$

, které jsou reprezentovány v třídě pravděpodobnostmi ω a rozdlíly μ. Rozdíl tříd $\mu_{0,1,T}(t)$ je

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0}\mu_1(t) = \sum_{i=t}^{L-1} i\frac{p(i)}{\omega_1}\mu_T = \sum_{i=0}^{L-1} ip(i) \tag{10}$$

Algoritmus probíhá následovně

1. Výpočet histogramu a pravděpodobnosti pro každou hodnotu intenzity
2. Prvotní nastavení $\omega_i(0)$ a $\mu_i(0)$
3. Provést pro všechny možné práhy $t = 1, ...$ maximální hodnoty

   Aktualizace ω_i a μ_i

   Výpočet $\sigma_b^2(t)$
4. Kýžený práh je maximum $\sigma_b^2(t)$

2.4 Detekce hran

Objekty v obraze (jsou-li vidět), se od svého okolí odlišují jinou hodnotou jasu, tudíž blízká hrana objektu je vyznačena vyšším jasovým rozdílem. V ideálním případě má objekt jednu hodnotu jasu a okolí jinou. Jenže tenhle případ se v reálném světě nestává (objekt je různě stínovaný, šum atp.), proto bylo předešlým úsilím jas objektu sjednotit a odlišit od okolí. Pokud bude objekt mít co největší jasový rozdíl oproti svému okolí, bude se nám lépe rozpoznávat hrany, které tento objekt popisují. Pro tuto detekci se používají hranové detektory a v následují části 2.4.2 si popíšeme nejjednodušší způsob detekce pomocí aproximace derivací a nejčastěji používaný hranový detektor Cannyho hranový detektor.

2.4.1 Detekce hran pomocí gradientu

Nejjednodušší detekcí hran je aplikace jednoho ze standartních hranových filtrů. Příklad konvolučních jader je níže, kde jsou vyobrazena jádra Prewittové11, Sobela12, Laplace v čtyř13 a

osmiokolí14. Tyto masky se přikládají na obraz, v následujících příkladech jsou matice pro detekci v ose x pro osu y se tyto matice transponují. Výpočtem aproximované derivace z okolních bodů zjistíme, zdali se jedná o hranu či nikoli, jedná-li se o hranu bude výsledná derivace velká.

$$P = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{11}$$

$$S = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{12}$$

$$L_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{13}$$

$$L_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{14}$$

2.4.2 Cannyho hranový detektor

Jak již bylo řečeno jedná se o nejpoužívanější hranový detektor ve zpracování obrazu, se kterým přišel pan J. F. Canny [1] na začátku osmdesátých let minulého století. Výsledkem jeho práce bylo stanovení určitých pravidel, které je vhodné splňovat k dosažení co nejlepšího výsledku. Za co nejlepší výsledek bychom mohli považovat detektor, který má minimální chybovost (detekování opravdových hran a nedetekování falešných hran), správnou lokalizaci hrany (hrana musí být detekována co nejblíže místa na kterém se opravdu nachází) a jeho jednoznačnost (nesmí docházet ke zdvojení hran)[2].
Základní 4 kroky Cannyho detektoru:

1. Odfiltrování šumu. Pro tento účel je použito Gaussovy filtrace. Příklad Gaussova jádra o $velikosti = 5$, který může být použit je na obrázku níže 15

$$G = \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{15}$$

2. Určení gradientu intenzity, podobně jako při konvoluci.

(a) Aplikujeme Sobelovu masku v $x16$ i $y17$ směrech

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{16}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{17}$$

(b) Nalezení velikost a směr gradientu

$$G = \sqrt{G_x^2 + G_y^2} \tag{18}$$

$$\theta = arctan(\frac{G_y}{G_x} \tag{19}$$

Směr gradientu je zaokrouhlen na $\frac{\pi}{4}$ úhlu

3. Z hodnot gradientů vytvořených předchozím krokem se odeberou body, které nejsou maximem. Tímto způsobem ztenčíme hranu - odebereme širší okolí a necháme pouze ty body, které mají největší gradient.

4. Prahováním s hysterézí, které již bylo popsáno dříve**??**. Pan Canny doporučil poměr prahů *horní* : *dolní* v rozmezí mezi 2 : 1 a 3 : 1.

2.4.3 Detekce segmentů čar

Line Segment Detector (Detektor segmentů čar) detektor pracující v lineárním čase a podávající přesnosti menší než velikost pixelu[8]. Je navržen aby fungoval na každém digitálním obrázku bez nutnosti přenastavování parametrů. Detektor si sám kontroluje falešné detekce a v průměru je povolena jedna na obraz.

Vstupem LSD je obraz ve stupních šedé a jako výstup vrací list detekovaných segmentů čar. V pseudokódu23 je ukázán princip. Obraz *Status* má stejnou velikost jako obraz v měřítku (*scaleImage*) a je využit k ukládání si pixelů, které již byly použity.
LSD byl navrhnut jako automatický nástroj analýzy obrazu, který nevyžaduje vnější přenastavování. Algoritmus je postaven na několika proměnných definujících jeho chování, které byly vhodně nastaveny pro fungování na všech obrazech. Algoritmus je naznačen na následujícím pseudokódu

Algorithm 1: LSD: Line Segment Detector

input: An image I.
output: A list *out* of rectangles.

$I_S \leftarrow \text{ScaleImage}(I, S, \sigma = \frac{\Sigma}{S})$
$(\text{LLA}, |\nabla I_S|, \text{OrderedListPixels}) \leftarrow \text{Gradient}(I_S)$
$\text{Status} \leftarrow \begin{cases} \text{USED}, & \text{pixels with } |\nabla I_S| \leq \rho \\ \text{NOT USED}, & \text{otherwise} \end{cases}$
foreach *pixel* $P \in$ OrderedListPixels **do**
  if Status$(P) =$ NOT USED **then**
    $region \leftarrow \text{RegionGrow}(P, \tau)$
    $rect \leftarrow \text{Rectangle}(region)$
    while $\text{AlignedPixelDensity}(rect, \tau) < D$ **do**
      $region \leftarrow \text{CutRegion}(region)$
      $rect \leftarrow \text{Rectangle}(region)$
    end
    $rect \leftarrow \text{ImproveRectangle}(rect)$
    $nfa \leftarrow \text{NFA}(rect)$
    if $nfa \leq \varepsilon$ **then**
      Add $rect \rightarrow out$
    end
  end
end

Obrázek 12: Pseudokód LSD algoritmu

2.5 Morfologické transformace

Morfologické transformace jsou jednoduché operace založené na tvaru obrazu[12]. Provádí se na binárním obrazu (obrazu složeného ze dvou jasových hodnot) a jako vstup se používá vybraný obraz určený pro transformaci a maska, která určuje povahu operace. Některé morfologické transformace jsou už v knihovně OpenCV[3] implementovány. Dvě základní operace jsou eroze a dilatace, které si následně popíšeme.

Obrázek 13: Vstupní obrázek morfologických transformací[12]

2.5.1 Eroze

Základní myšlenka eroze je podobná erozi půdy. Odstraňují se přečnívající části objektu v popředí. Maskou se prochází celý obraz (obdobně jako u 2D konvoluce), který nabývá dvou hodnot (například 0 nebo 1). Pixel v obraze na pozici středového pixelu masky bude nabývat hodnoty

1, pokud mají i všechny ostatní pixely pod maskou hodnotu 1, jinak se mu nastaví hodnota 0. Tudíž nastane to, že všechny pixely kolem hranice budou odstraněny podle velikosti masky a ve výsledku se zmenší i samotný objekt.

Obrázek 14: Výstup pro transformaci eroze[12]

2.5.2 Dilatace

Dilatace je v podstatě opak eroze. Pixel pod maskou bude nabývat hodnoty 1, pokud bude mít alespoň jeden pixel pod maskou hodnotu 1. Při odstraňování šumu se nejčastěji používá kombinace eroze následována dilatací, protože eroze nám s odstraněním šumu objekt zmenší, ale dilatace jej zase zvětší, a to už bez šumu, kterého se chceme zbavit. Tato metoda je také vhodná pro spojování rozpojených hran.

Obrázek 15: Výstup pro transformaci dilatace[12]

2.5.3 Morfologický gradient

Další zajímavou transformací je morfologický gradient, jenž má za výsledek rozdíl mezi dilatací a erozí obrazu.

Obrázek 16: Výstup pro transformaci morfologického gradientu[12]

3 Implementace

Pro implementaci jsem si zvolil jazyk C++ s použitím vývojového prostředí Microsoft Visual Studio 2013. C++ jsem si zvolil protože je nejlépe podporován knihovnou OpenCV[3]. Pro grafické uživatelské rozhraní jsem využil vývojového prostředí Qt, které jsem pomocí addonu přidal do Visual Studia.

3.1 OpenCV

Ke zpracování obrazu jsem využil knihovny OpenCV[3], která je šířena pod BSD licencí, tudíž je volně k využití pro osobní, akademické, ale i komerční účely. Knihovna podporuje jazyky C, C++, Python a Java. Je multiplatformní a beží na Windows, Linux, Mac OS, iOS i Androidu. Knihovna obsahuje mnoho tříd usnadňující práci s obrazy jako je matice Mat, funkce $imread()$ apod. Dále je její výhodou, že obsahuje již mnoho implementovaných funkcí pro zpracování obrazu, jako je funkce pro prahování $threshold()$ nebo pro Cannyho hranový detektor $Canny()$. Knihovna se skládá z mnoha modulů, zde uvádím jejich krátký výčet

- *core* - základní funkcionality a datové struktury, například matice Mat
- *imgproc* - část zabývající se zpracováním obrazu, jako jsou filtrace obrazu, histogramy či geometrické transformace
- $features2d$ - detektory příznaků, deskriptory a jejich vyhodnocování
- *gpu* - algoritmy pro rychlejší zpracování na grafické kartě (GPU) pro ostatní moduly knihovny

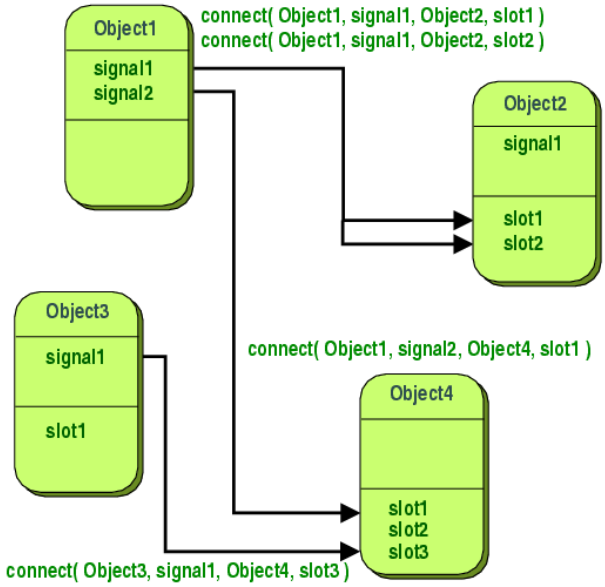
3.2 Qt

Qt je multiplatformní framework vytvořen roku 1999 společností Trolltech[5]. Aplikace napsané pomocí tohoto framworku je možno šířit pod licencí GPL, LGPL, nebo, po splnění podmínek, komerčně. Qt je knihovna programovacího jazyka C++, ale existuje i pro další jazyky. Výhodou je například Qt Designer, který je jednoduchý na vytváření grafického rozhraní. Příkladem využití tohoto frameworku může být Skype, Google Earth, VirtualBox či Opera. Zde uvádím využité moduly[7] knihovny v této práci

- $QtCore$ - základ negrafických tříd využívaný v ostatních modulech
- $QtGUI$ - třídy pro grafické uživatelské rozhraní, obsahuje i OpenGL
- $QtWidgets$ - třídy pro rozšíření $QtGUI$ widgety $C++$

3.2.1 Signály a sloty

Důležitou funkcionalitou oproti klasickému C++ je možnost použití signálů a slotů, které jsou využity pro komunikaci mezi objekty[6]. V GUI je často potřeba, aby jedna komponenta informovala druhou, například když uživatel klikne na tlačítko *Otevřítsoubor*, pak se očekává zobrazení dialogu pro výběr souboru. V Qt můžeme jednomu objektu definovat signál, který bude emitovat a v jiném objektu definovat *slot*, což je funkce, která se zavolá při emitovaném signálu. Signálům můžeme určit jaké argumenty budou zasílat a *sloty* přijímat. Funkcí *connect*() určíme, který signál má spustit danou funkci.



Obrázek 17: Abstraktní schéma propojení pomocí signálů a slotů[6]

Tento mechanismus je typově bezpečný, signál musí obsahovat stejné argumenty jako slot, respektive slot může mít méně argumentů než signál, protože dokáže ignorovat argumenty navíc. Díky této vlastnosti se dokážeme vyvarovat chybám již v kompilační části. Když je signál vysílán, tak nikdy neví, jaké sloty jej přijmou. Tato mechanika Qt zajišťuje, že když je signál a slot propojen, tak se slot zavolá vždy ve správnou chvíli, kdy byl signál vyslán i s jeho parametry. Signály a sloty mohou vzít jako parametr jakýkoli typ a jakékoli množství. Všechny třídy dědící z *QObject* nebo z jeho podtříd (například *QWidget*) mohou obsahovat signály a sloty. Signály jsou vysílány objekty, které změnily svůj stav způsobem, o kterém by měly být ostatní objekty informovány. Toto je dostačující ke komunikaci mezi objekty. Vysílající objekt nepotřebuje vědět zdali jeho signál nějaký objekt přijme. Sloty mohou být použity pro zachytávání signálů, ale také se mohou použít jako normální funkce. Podobně jako objekt neví, jestli někdo obdrží signál, tak slot neví, jestli má propojený nějaký objekt, který by mu vysílal signály. Velkou výhodou tohoto je jistá míra nezávislosti. Je možno připojit jakékoli množství signálů k jednomu slotu a zároveň

můžeme připojit jakékoli množství slotů na jeden signál. Pospolu signály a sloty fungují jako mocný nástroj při programování[6].

Příklad

Mějme následující třídu

```
1  class Counter
2  {
3  public:
4      Counter() { m_value = 0; }
5
6      int value() const { return m_value; }
7      void setValue(int value);
8
9  private:
10     int m_value;
11 };
```

Výpis 1: Třída Counter

Pokud z ní uděláme *QObject*, který bude obsahovat signál a slot, tak by mohla vypadat následovně

```
1  #include <QObject>
2
3  class Counter : public QObject
4  {
5      Q_OBJECT
6
7  public:
8      Counter() { m_value = 0; }
9
10     int value() const { return m_value; }
11
12 public slots:
13     void setValue(int value);
14
15 signals:
16     void valueChanged(int newValue);
17
18 private:
19     int m_value;
20 };
```

Výpis 2: Třída Counter dědící z *QObject*

QObject verze má stejné vnitřní proměnné, ale navíc obsahuje komponenty pro použití signálů a slotů. Pomocí emitování signálu *valueChanged*() může dávat ostatním třídám na vědomí, že se její hodnota změnila. Taktéž má navíc slot *setValue*(), kterým může zachytávat signály ostatních tříd. Všechny třídy, které chtějí využívat této techniky, musí obsahovat *Q_OBJECT* na vrchu své deklarace a taktéž dědit z třídy *QObject*, ať už přímo či nepřímo.
Sloty jsou implementovány programátorem, podobně jako funkce, zde je možná implementace slotu pro třídu výše.

```
1 void Counter::setValue(int value)
2 {
3     if (value != m_value) {
4         m_value = value;
5         emit valueChanged(value);
6     }
7 }
```

Výpis 3: Slot *setValue*()

Řádek *emitvalueChanged*(*value*); vysílá signál z objektu s novou hodnotou proměnné v argumentu. Následně si vytvoříme dva objekty *Counter* a propojíme signál *valueChanged*() prvního objektu se slotem *setValue*() druhého objektu pomocí funkce *connect*()

```
1     Counter a, b;
2     QObject::connect(&a, SIGNAL(valueChanged(int)),
3                      &b, SLOT(setValue(int)));
4
5     a.setValue(12);   // a.value() == 12, b.value() == 12
6     b.setValue(48);   // a.value() == 12, b.value() == 48
```

Výpis 4: Ukázka propojení signálu a slotu

a.setValue(12) vyšlet signál *valueChanged*(12), který objekt *b* zachytí ve svém *setValue*() slotu. Pak objekt *b* vyšle stejný signál *valueChanged*(), ale jelikož k němu nebyl připojen žádný slot, tak bude ignorován. V příkladu je podmínka $if(value! = m\_value)$, aby nedocházelo k zacyklení.

Ve výchozím nastavení se z jednoho objektu vysílá tolik signálů, kolik bylo na něj připojených slotů, potažmo pro dvě připojení se vyšlou dva duplicitní signály. Tyto signály se dají zrušit zavoláním funkce *disconnect*(). Při nastavení parametru $Qt :: UniqueConnection$ se vyšle signál

jenom v případě neexistujícího stejného signálu. Při pokusu o vyslání stejného signálu vrátí funkce $false$.

3.3 Struktura aplikace

Vytvořil jsem třídu QT_Weld, která v podstatě obsahuje většinu funkcionality. Tato třída je potomkem Qt třídy $QMainWindow$, která se používá pro vytvoření GUI a stará se o rozložení objektů v okně aplikace. Pomocí $Qtdesigner$ jsem vytvořil uživatelské rozhraní, které obsahuje hlavní okno pro zobrazování vstupního obrazu a možností stanovit body, kterými se měří parametry výrobku. Nad hlavním oknem se nachází tlačítko pro otevření souboru a tlačítka pro všechna měření parametrů, která vyžadují uživatelem stanovené body. V QT_Weld je obsažen celý proces předzpracování obrazu, protínání hran přímkou, detekce rohových bodů aj.

Oknem pro zobrazování obrazu je třída $myGraphics$, která dědí z $QGraphicsView$, jež obstarává základní funkcionality pro zobrazení obrazu, přidávání bodů, zoom. Také emituje signály pro akce myši nad ní, které dále zpracovávám, například při určení pozice v obrazu, pro umístění bodu zájmu.

3.3.1 Body zájmu

Nejdůležitější částí jsou třídy pro jednotlivé body měření parametrů. Pro všechny jsem vytvořil generického předka, který dědí z třídy $QGraphicsItem$ a taky z $QObject$. Pojmenoval jsem ji $PointOfInterest$ a obsahuje základní funkce pro každý bod, jako jsou nastavení bodu dle pozice kliknutí v okně $setImagePoint()$, nastavení velikosti čtvercového obalu $boundingRect()$ či tvaru bodu ve funkci předka $paint()$5. Tato třída se stará o překreslení i určení přesné pozice v obraze, také o komunikaci s nadřazenou vrstvou $myGraphics$, které předává signály.

```
1  void PointOfInterest::paint(QPainter *painter, const QStyleOptionGraphicsItem *
       option, QWidget *widget)
2  {
3     QPen pen;
4     if (Pressed)
5     {
6        pen = QPen(Qt::red, 3);
7     }
8     else
9     {
10       pen = crossPen;
11    }
12    painter->setPen(pen);
13    // Vykresleni krize ve tvaru +
```

```
14      painter->drawLine(25,0,25,50);
15      painter->drawLine(0,25,50,25);
16  }
```

Výpis 5: Funkce *paint*() pro vykreslení bodu

Každý jednotlivý bod parametru má ovšem svá specifika, jako je třeba jeho barva, či emitování signálu, který volá jeho funkci (slot), anebo výpočet jeho nové pozice.

CornerPoint - Pro bod, který určuje hranu svaru jsem vytvořil třídu s názvem $CornerPoint$, tyto body se v aplikaci nacházejí právě čtyři, jelikož přechod svar-plech je dán právě čtyřmi body. Tyto body jsou v aplikaci vyznačeny černou barvou a emitují signál $cornerPointMovedSignal()$ při akci puštění levého tlačítka myši $mouseReleaseEvent()$. Tento signál zavolá funkci $computeAll()$, která znovu hrany plechů aproximuje přímkou, přepočítá úhly mezi plechy, tloušťky materiálu, lineární a radiální přesazení apod., zkrátka přepočte všechny parametry, které posun bodu může ovlivnit.

HeightPoint - Pro měření parametru nadměrného převýšení svaru jsem vytvořil třídu $HeightPoint$, která je v aplikaci reprezentována tmavě fialovou barvou. Tyto body můžou být v aplikaci až dva, jeden pro měření horního a jeden pro měření dolního převýšení. Bod emituje signál $measureHeightSignal()$, kdykoliv se posune anebo umístí. Tento signál spustí proces přepočtu výšek převýšení ve funkci $measureHeight()$.

InflammationPoint - Tento bod se využívá k měření parametru souvislého zápalu. Tyto body mohou být v aplikaci až čtyři, jelikož může být zápal až na čtyřech místech, a je znázorněn tmavě žlutou barvou. Bod $InflammationPoint$ má navíc id, které reprezentuje hranu u které je nejblíže. Tyto body při umístění emitují signál $inflammationPointMoveSignal(id)$, který posílá své id pro identifikaci pozice rohu svaru. Dle tohoto identifikátoru se přepočte pouze korespondující parametr. Například pro levý horní zápal má index $UP\_LEFT\_CORNER6$, proto se přepočte jenom tento parametr.

```
const int UP_LEFT_CORNER = 0;
```

Výpis 6: Globální konstanta pro levý horní roh

ScalePoint - Tyto dva body jsou o trochu jiné než ostatní a reprezentují měřítko. Jedná se o vzdálenost mezi nimi podělená hodnotou v políčku měřítka. Vyznačuje se modrou a skládá se ze dvou bodů. Jedním bodem se nastaví počáteční bod měřítka a druhým koncový, tyto body se následně spojí šedou úsečkou. Při puštění bodu na místě se emituje signál $scalePointMovedSignal()$, který signalizuje, že se měřítko změnilo a následně se pomocí funkce $computeAll()$ přepočtou všechny hodnoty, dle nově stanoveného měřítka.

WeldPoint - Jedná se o body reprezentující měření strmého přechodu svaru, kde se mohou vyskytovat až čtyři body. Tyto jsou v aplikaci znázorněny zelenou barvou a při každém puštění tlačítka myši se přesunou na nejbližší bod svaru a emituje se signál *measureWeldAngleSignal*(), který zajistí spočtení daného úhlu mezi úsečkou spojující tento bod s nejbližší bodem *CornerPoint* a úsečkou reprezentující přilehlou hranu plechu. *WeldPoint* si uchovává seznam bodů, které reprezentují hrany svaru, v seznamu *points*. Algoritmus vyhledání nejbližšího bodu vypadá následovně

```
1  void WeldPoint::getClosesPoint(){
2     QPointF myPos = getImagePoint();
3     Point2f actualPos(myPos.x(), myPos.y());
4     float minDistance = 9999999;
5     Point2f minPoint;
6     for (auto point: points){
7        float euclid = norm(actualPos-point);
8        if (euclid < minDistance){
9           minDistance = euclid;
10          minPoint = point;
11       }
12    }
13    setImagePoint(minPoint.x, minPoint.y);
14 }
```

Výpis 7: Globální konstanta pro levý horní roh

BeadPoint - Bod, kterým se měří výška svarové housenky, vyznačuje se tmavě tyrkysovou barvou a v aplikaci se mohou nacházet maximálně dva body najednou. Tento bod emituje signál *measureBeadSignal*(), který má slot *beadPointSet*(), tento slot se zasluhuje o přepočítání parametru výšky svarové housenky pomocí funkce *minBead*().

DepressionPoint - Tento bod se vyznačuje tmavě modrou barvou slouží k měření parametru prolákliny. Hloubka prolákliny se měří až na dvou místech v obraze. Tento bod emituje signál *measureDepressionSignal*(), který zachytává slot *depressionPointSet*(). Tento slot následně určí zdali se jedná o horní nebo dolní proláklinu, dle polohy bodu a vypočte její velikost pomocí funkce *maxDepression*().

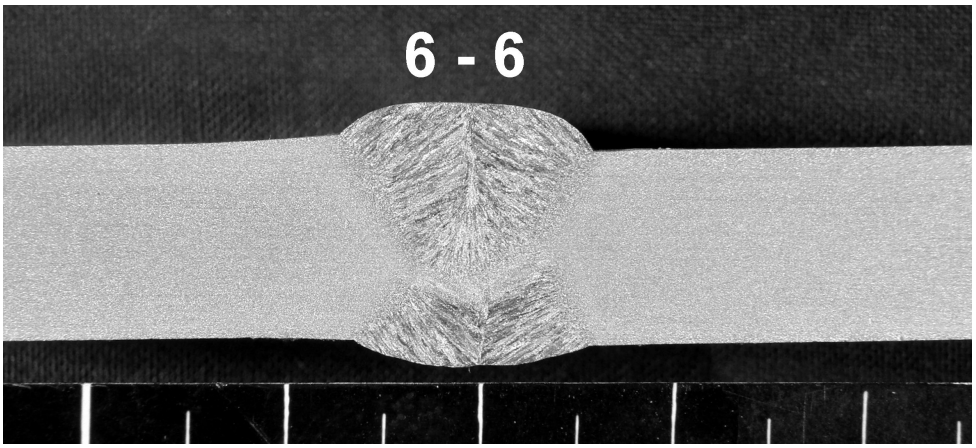
3.4 Načtení vstupního obrazu

Pro načtení vstupního obrazu18 jsem využil funkce knihovny OpenCV[3] *imread*() v následující syntaxi

```
Mat src = imread(filename, IMREAD_GRAYSCALE);
```

Výpis 8: Použití bilaterálního filtru knihovny OpenCV[3]

jelikož povaha vstupního obrazu a následujícího zpracování nejlépe odpovídá obrazu ve stupních šedi, využil jsem parametru $IMREAD\_GRAYSCALE$.



Obrázek 18: Vstupní obraz

3.5 Předzpracování

V první řadě je důležité předzpracování vstupních obrazů, jelikož jsou obrázky nedokonalé, zašuměné a často i dost odlišné. Na začátku jsem zkoušel konvoluce2.2.1 s různými maskami a jejich velikostmi. Při použití menší masky $3x3$ zůstávalo hodně zašuměných míst, naopak při použití větší masky $15x15$ docházelo k rozmazání hran a Cannyho hranový detektor měl problém detekovat skutečné hrany. V OpenCV je implementována funkce pro rozmazání Gaussianovou maskou $GaussianBlur()$, která byla vhodná pro otestování rozmazání. Proto bylo důležité najít způsob jak rozmazat šum, ale zároveň zachovat hrany, což právě umožňuje v aplikaci použitý bilaterální filtr2.2.2.
Pro použití tohoto filtru jsem využil funkce

```
1   bilateralFilter(src, dst, kSize, sigmaX, sigmaY);
```

Výpis 9: Funkce bilaterálního filtru knihovny OpenCV[3]

Parametry funkce jsou:

1. src - vstupní matice obrazu, v mém případě ve stupních šedé, což je odpovídající typ CV_8UC1
2. dst - výstupní matice obrazu se stejnou velikostí a typem jako *src*
3. kSize - velikost Gaussovy matice masky, která může mít jinou šířku a výšku, ale obě hodnoty musí být kladné a liché
4. sigmaColor - Filtrace sigmy v barevném prostoru. Větší hodnota znamená, že větší rozdíly barvy budou ve výsledku smíchány, výsledkem budou větší plochy

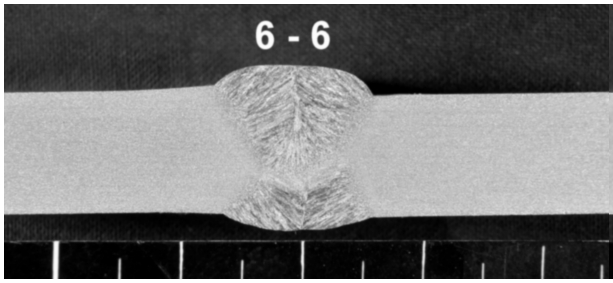
5. sigmaSpace - standartní deviace Gaussovy masky v Y směru

Metodou "pokus-omyl"jsem došel k závěru, že na mých vstupních vzorcích je nejvíce vyhovující následující nastavení bilaterálního filtru

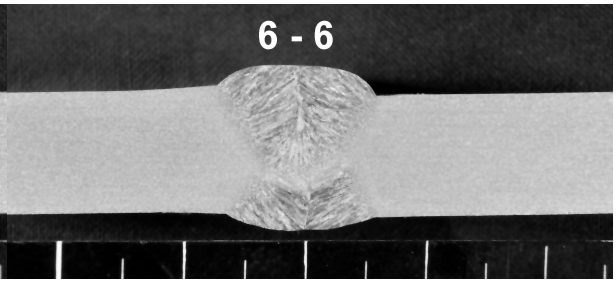
```
1   bilateralFilter(src, dst, 15, 55, 55);
```

Výpis 10: Použité nastavení bilaterálního filtru

Na následujících obrázcích jde vidět rozdíl mezi použitím klasické konvoluce Gaussovou maskou19 a přidáním vzdálenostního parametru20 (použitím bilaterálního filtru).



Obrázek 19: Konvoluce Gaussiánem

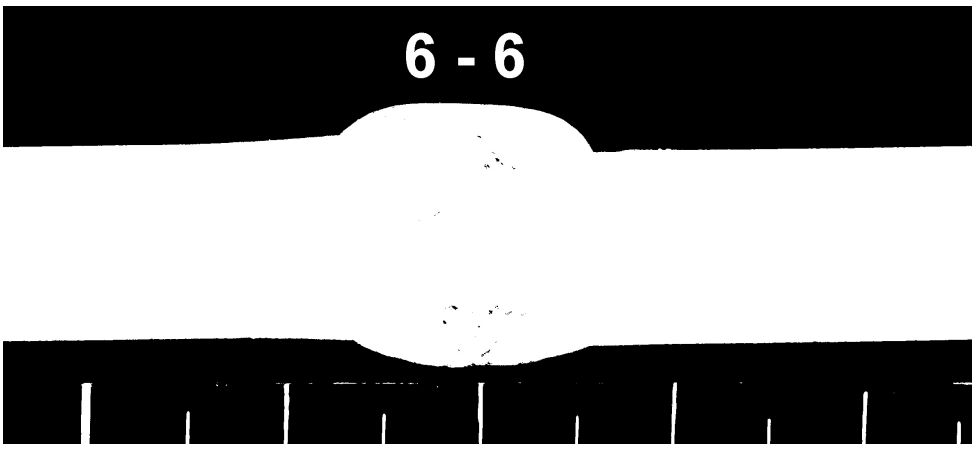


Obrázek 20: Bilaterální filtr

Obrázek rozostřený bilaterálním filtrem jsem upravil prahováním s použitím Otsuova algoritmu2.3.2, čímž jsem obraz ve stupních šedé převedl na binární obraz (obraz skládající se z pouze dvou hodnot jasu). Na výsledném obrazu21, po prahování, je daleko jednodušší detekovat správně hrany. Pro prahování jsem použil následující funkci z OpenCV[3]

```
1   threshold(src, dst, 0, 255, THRESH_BINARY | THRESH_OTSU);
```

Výpis 11: Použité nastavení prahování



Obrázek 21: Binarizovaný obraz pomocí Otsuova prahování

3.6 Detekce hran

K detekci hran jsem použil již implementované metody *Canny*() z knihovny OpenCV[3] pro Cannyho hranový detektor. Tento detektor jsem si vybral pro všeobecně dobré výsledky při různých podmínkách. Cannyho detektor z knihovny OpenCV se používá následujícím způsobem

```
Canny(src, edges, threshold_L, threshold_L, kSize, L2gradient);
```

Výpis 12: Funkce Cannyho detektoru

Parametry funkce jsou:

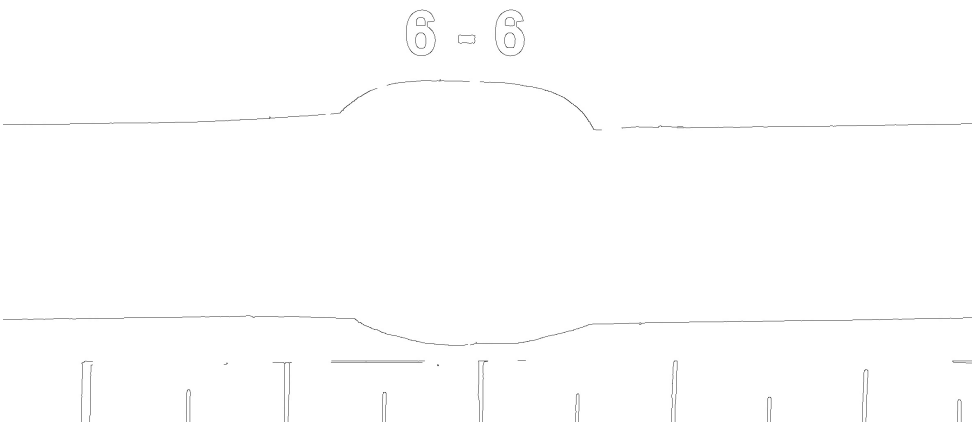
1. src - vstupní matice obrazu, v mém případě výstup obrazu upraveném bilaterálním filtrem2.2.2
2. edges - výstupní matice obrazu s již detekovanými hranami
3. thresholdL - spodní práh pro hysterézní prahování2.3.1
4. thresholdH - horní práh pro hysterézní prahování2.3.1
5. kSize - velikost Sobelovy matice masky12
6. L2gradient - nastavení $(true/false)$ pro použití přesnější normy $L_2 = \sqrt{(dI/dx)^2 + (dI/dy)^2}$ k výpočtu velikosti gradientu obrazu $(L2gradient = true)$.

Nejlépe se mi osvědčilo následující nastavení

```
Canny(src, dst, 200, 400, 3, true);
```

Výpis 13: Použité nastavení Cannyho detektoru

Na obrázku22 je viditelný výstup hranového detektoru pro výše zmíněný vstupní obraz18.



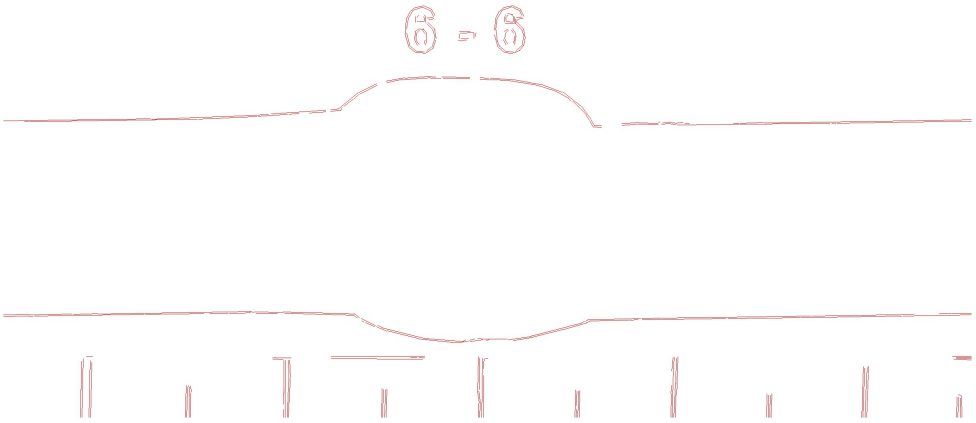
Obrázek 22: Výsledný obraz po aplikaci Cannyho hranového detektoru na vstupní obraz upravený bilaterálním filtrem

3.7 Vytvoření segmentů čar

V dalším zpracováním obrazu jsem využil funkcí z knihovny OpenCV pro detekci segmentů čar[4]. Jako vstup dobře posloužil obraz s detekovanými hranami, což byl výstup funkce *Canny*(). Prvně bylo třeba na paměti vytvořit *LineSegmentDetector*, detekovat úsečky na matici výstupu Cannyho detektoru a pro ověření si nechat úsečky vykreslit do matice23 s následujícím nastavením

```
1   LineSegmentDetector lsd = createLineSegmentDetector(LSD_REFINE_ADV);
2   userdata->ls->detect(src, *userdata->lines);
3   userdata->ls->drawSegments(src, *userdata->lines);
```

Výpis 14: Funkce pro vytvoření detektoru čar



Obrázek 23: Detekované úsečky čar

3.8 Detekce přechodu svaru

Aplikace má za úkol automaticky detekovat místa na hraně, ve kterých plát plechu přechází ve svar. Tato část je důležitá pro urychlení práce zaměstnance zabývajícího se kontrolou kvality. Zkontroloval jsem si, jak vstupní obrazy vypadají. Na všech fotografiích svařených kovových plátů, jsou pláty vodorovně. Z čehož jsem vycházel v následujícím algoritmu.

V obraze s množstvím úseček reprezentující hrany, hledám dvě místa, kde se jich vyskytuje nejvíce a jsou zároveň ve vodorovném směru. Obraz jsem si rozdělil dle výšky po pěti procentech jeho absolutní výšky. Najdu-li dvě tato místa, prohlásím je za přibližné polohy vrchní a spodní hrany plátů. Z těchto dvou míst si stanovím střed výrobku a rozdělím body úseček do příslušných kvadrantů, viz následující úryvek kódu funkce *detectPosition*()15

```
    void detectPosition(Data* data){
    vector<Point2f> histogram[20];
    vector<int> significantPointHistogram = vector<int>(20);
    float height = data->image->rows;
    for (int i = 0; i < data->lines->size(); i++)
    {
        Point2f p1 = Point2f(data->lines->at(i)[0], data->lines->at(i)[1]);
        Point2f p2 = Point2f(data->lines->at(i)[2], data->lines->at(i)[3]);

        data->points.push_back(p1);
        data->points.push_back(p2);

        float histIndexp1 = p1.y / height * 20;
        float histIndexp2 = p2.y / height * 20;

        histogram[(int)histIndexp1].push_back(p1);
        histogram[(int)histIndexp2].push_back(p2);

        //skip small lines
        if (abs(p1.x - p2.x) < 50)continue;

        significantPointHistogram[(int)histIndexp1]++;
        significantPointHistogram[(int)histIndexp2]++;
    }

    auto max = std::max_element(significantPointHistogram.begin(), significantPointHistogram.end());
    int plate1index = std::distance(significantPointHistogram.begin(), max);
    significantPointHistogram[plate1index] = -1;

    auto secondMax = std::max_element(significantPointHistogram.begin(), significantPointHistogram.end());
    int plate2index = std::distance(significantPointHistogram.begin(), secondMax);

    vector<Point2f> points = histogram[plate1index];
    points.insert(points.end(), histogram[plate2index].begin(), histogram[plate2index].end());

should be in the middle of weld
    Point2f avgPoint;
    for (int i = 0; i < points.size(); i++)
    {
        avgPoint += points.at(i);
    }
    avgPoint.x = avgPoint.x / points.size();
    avgPoint.y = avgPoint.y / points.size();
    cross(*data->image, avgPoint);
    data->avgPoint = avgPoint;
    for (Point2f p : points){

        bool top = p.y < avgPoint.y;
        bool bottom = !top;
        bool left = p.x < avgPoint.x;
        bool right = !left;
        int quardIndex = 0;
        if (top && left)
        {
            quardIndex = 0;
        }
        else if (top && right)
        {
            quardIndex = 1;
        }
        else if (bottom && right)
        {
            quardIndex = 2;
        }
        else if (bottom && left)
        {
            quardIndex = 3;
        }
        data->quadr_points[quardIndex].push_back(p);
    }
    return;
}
```

Výpis 15: Detekce vrchní a spodní hrany plátů

Mám-li obraz rozdělen na kvadranty a v každém kvadrantu mám příslušné body hran materiálu, přejdu ke kroku, kdy si ze všech bodů v kvadrantu vypočtu vektor hrany plechu funkcí *detectPlates*(). Využil jsem funkce z knihovny OpenCV[3], která má následující argumenty

```
fitLine(points, lines, distType, apram, reps, aeps);
```

Výpis 16: Proložení listu bodů přímkou

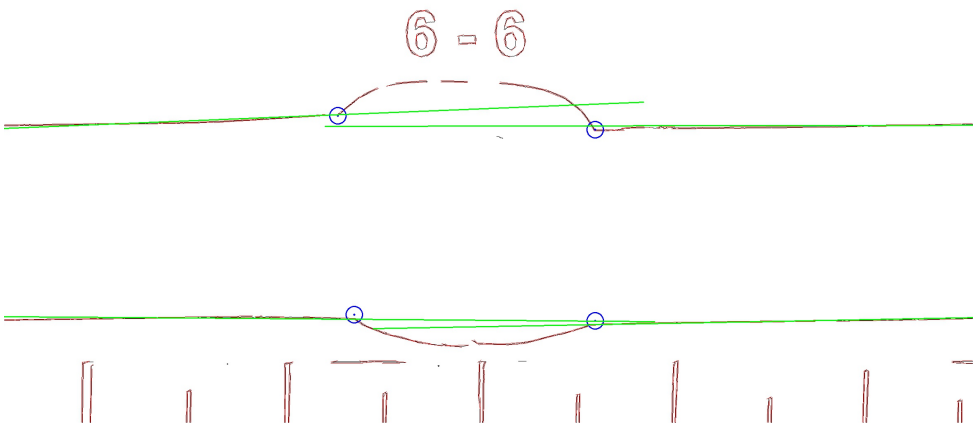
Argumenty funkce

- *points* - vstupní list bodů
- *lines* - výstupní list normalizovaných vektorů (ve 2D) typu $Vec4f$
- *distType* - vzdálenost použita M-estimátorem
- *param* - číselný parametr pro některé typy vzdáleností. Pokud je 0, bude použita optimální hodnota
- *reps* - rádiusová přesnost (vzdálenost mezi původním místem a čárou)
- *aeps* - přesnost pro úhel

Mě se osvědčilo toto nastavení

```
fitLine(points, lines, CV_DIST_L2, 0, 0.01, 0.01);
```

Výpis 17: Použité nastavení $fitLine()$



Obrázek 24: Výsledek aproximovaných přímek hran pomocí $fitLine()$

3.9 Detekce hran svaru

Na těchto aproximovaných hranách se snažím detekovat správné pozice přechodu mezi svarem a plechem. Tyto body jsou důležité pro správnost téměř všech měření, jelikož z jedné strany

ohraničují hranu plechu a tím zajišťují, že do výpočtu aproximované přímky hrany plechu nebudou započteny i detekované body na svaru. Tuto detekci zajišťuje funkce mnou implementovaná *detectCorners*()**??**, kde si pro každý kvadrant seřadím body, které odpovídají hranám směrem od hrany obrazu do středu. Následně porovnávám kolmé vzdálenosti k přímce hrany a pokud je bod blízko, tedy je stále na hraně plechu a zároveň je nejvíce ve středu svaru, tak jej prohlásím za rohový bod. Tyto body *CornerPoint* si ukládám ve struktuře *Data*. Když mám detekovaný rohový bod, pak zpřesním výpočet přímky pouze na body, které jsou od okraje obrazu, po tento rohový bod. Tímto je předzpracování a automatická detekce rohového bodu u konce. Tyto body je obtížné detekovat správně, protože obrázky svarů můžou mít různé deformace před nebo i přímo na samotném svaru. Může vznikat zápal i proláklina. Z některých obrázků je i pro člověka, ne úplně znalého, těžké posoudit, kde správně svar začíná. Tudíž přesná implementace této detekce je takřka nemožná.

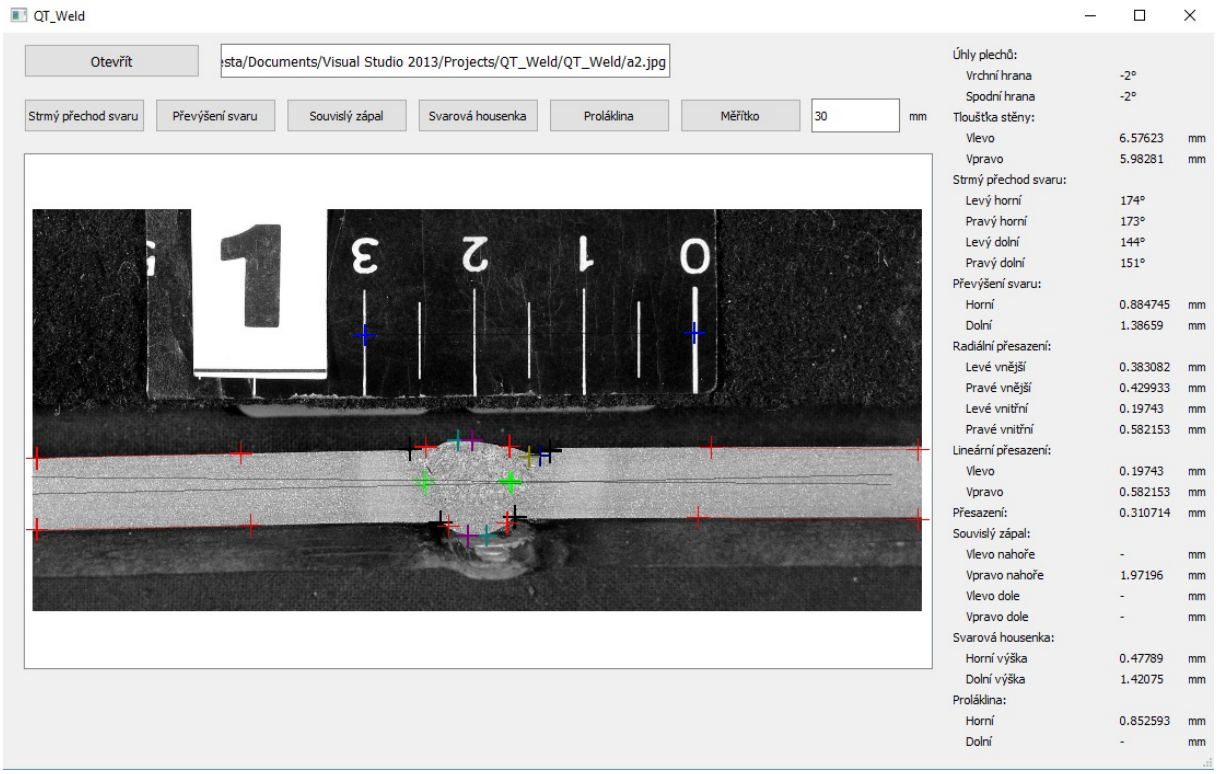
Jako možné vylepšení by se při větším počtu vstupních obrazů dalo implementovat strojové učení. Díky neuronovým sítím, které se dokáží učit na podněty učitele, by se dalo relativně snadno toto implementovat, myslím si, že výsledky by byly o dost přesnější, jelikož každý vstupní obrázek je něčím specifický a navzájem jsou také často dosti odlišné.

```
1  void QT_Weld::detectCorners(){
2
3      for (auto& corner : userdata->corners){
4          scene.removeItem(corner);
5      }
6      userdata->corners.clear();
7
8      for (int i = 0; i < 4; i++){
9
10         vector<Point2f> points = userdata->quadr_points[i];
11
12         std::vector<double> pDist;
13
14         // plates sort - middle points are those with the bigest index
15         if (i == 0 || i == 3)
16             std::sort(points.begin(), points.end(), xsortL);
17         else
18             std::sort(points.begin(), points.end(), xsortR);
19
20         for (int p = 0; p < points.size() - 1; p++){
21             Point2f p1 = Point2f(userdata->plates->at(i)[0], userdata->plates->at(i)[1]);
22             Point2f p2 = Point2f(userdata->plates->at(i)[2], userdata->plates->at(i)[3]);
23             double d = perpendicularDistance(p1, p2, points.at(p));
24             pDist.push_back(d);
25         }
26         double threshold = 0;
27         int size = pDist.size() * 1 / 2;
28         for (int p = 0; p < size; p++){
29             threshold += pDist.at(p);
30         }
31         threshold /= size;
32         for (int p = pDist.size() - 1; p > 5; p--){
33             double sumD = pDist.at(p - 4) + pDist.at(p - 3) + pDist.at(p - 2) + pDist.at(p - 1) + pDist.at(p);
34             if (sumD / 5 < threshold || abs(pDist.at(p)) < 10 || p == 6){//p==6 - last iteration
35                 CornerPoint *cp = new CornerPoint();
36                 scene.addItem(cp);
37                 userdata->corners.push_back(cp);
38                 cp->setImagePoint(points.at(p).x, points.at(p).y);
39                 break;
40             }
41         }
42     }
43
44 }
```

Výpis 18: Funkce pro detekci rohů svaru *detectCorner*()

3.10 Grafické uživatelské rozhraní

Na obrázku níže je vyobrazena výsledná podoba okna, které se otevře při spuštění aplikace. Uživatel si tlačítkem vlevo nahoře otevře vstupní obraz, na kterém chce provádět měření kvalitativních parametrů. V okně se zobrazí načtený obraz s již detekovanými parametry jako jsou rohy svarů, či hrany plechů. Jelikož detekce není vždy stoprocentní, tak má uživatel možnost si tyto prvky posunout dle libosti. Na řádku níže jsou všechna tlačítka, která vyžadují uživatelské zadání bodů pro měření parametrů. Jako poslední tlačítko je **Měřítko**, kterým si uživatel nastaví rozpětí měřítka a v okénku za ním udá měřenou vzdálenost. Měřítko se používá u všech délkových měření, aby byly výsledné hodnoty dle reality. Pod tlačítky je okno zobrazující obraz a všechny potřebné body, toto okno je přibližovatelné pro přesnější určení bodů. Na pravé straně jsou vyhodnoceny všechny měřené parametry.



Obrázek 25: Grafické uživatelské rozhraní aplikace

4 Měřené parametry

Tato aplikace umožňuje měření některých parametrů výsledného produktu, které důležité pro výstupní kontrolu kvality. Při výrobním procesu je nutno dodržovat normy a produkovat kvalitní výrobky. V následujících kapitolách budou popsána jednotlivá měření parametrů a porovnání mezi ručním měřením a měřením aplikace. Ruční měření na fotografiích mi přišlo dost nepřesné a hlavně zdlouhavé.

4.1 Měření úhlů mezi hranami plechů

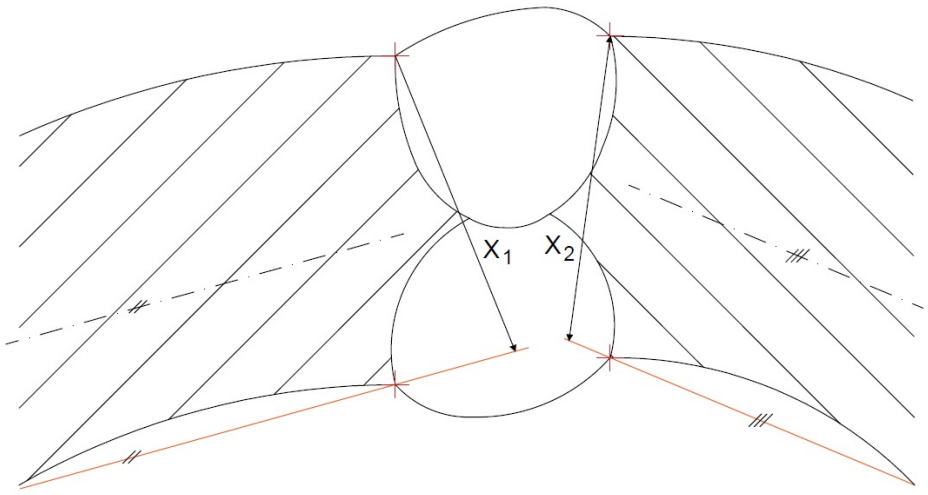
Aplikace měří úhly protilehlých hran plechů a to tak, že se každou hranou aproximuje přímka, která ji reprezentuje. Pak se vypočte úhel mezi protilehlými hranami (levou horní a pravou horní nebo levou dolní a pravou dolní). Porovnání naměřených hodnot je v tabulce níže

Tabulka 1: Naměřené hodnoty úhlů protilehlých stěn

	Horní			Dolní		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
1	2	2	0	2	1	1
2	1	2	1	1	2	1
3	1	2	1	1	2	1
4	2	1	1	0	1	1
5	2	1	1	2	1	1
6	2	1	1	1	1	0
7	6	6	0	6	6	0
8	1	0	1	2	1	1
9	2	1	1	2	1	1
10	2	2	0	2	1	1
11	2	2	0	0	1	1

4.2 Tloušťka stěny

Měření tloušťky stěny dle normy EN ISO 3183 probíhá následujícím způsobem. Tloušťka stěny se počítá na levé x_1 a pravé straně x_2. Levá strana se měří jako kolmá vzdálenost levého horního okrajového bodu svaru a dolní levou přímkou materiálu. Pravá strana se měří obdobně, tudíž jako kolmá vzdálenost pravého horního okrajového bodu svaru a dolní pravou přímkou materiálu. Náčrt měření je níže26. Porovnání naměřených hodnot je v tabulce pod obrázkem2.



Obrázek 26: Nákres měření tloušťky stěny

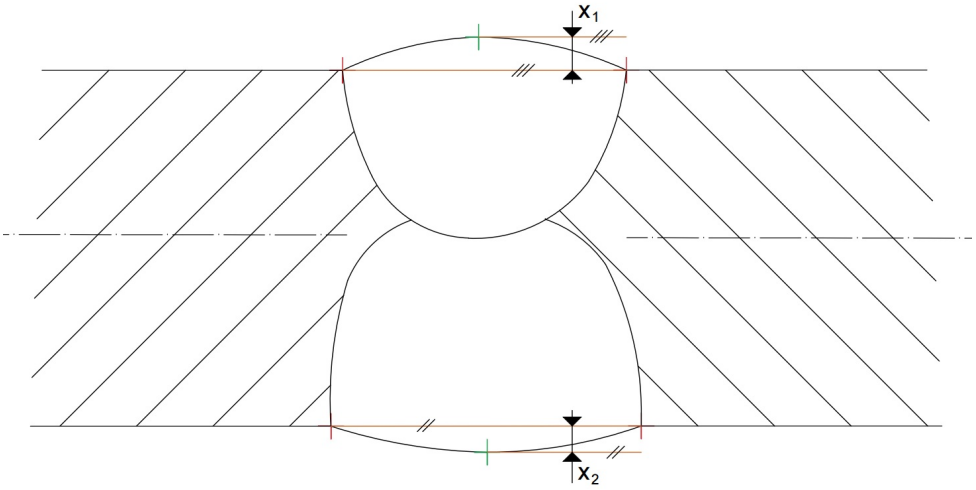
Tabulka naměřených hodnot

Tabulka 2: Naměřené hodnoty tloušťky stěny

	Levá strana			Pravá strana		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
1	10,52	10,31	0,21	10,00	9,96	0,04
2	6,67	6,76	0,09	6,67	6,21	0,46
3	6,00	6,39	0,39	6,00	6,03	0,03
4	2,76	3,14	0,38	2,89	2,95	0,06
5	4,92	5,10	0,18	5,16	5,26	0,11
6	2,98	3,61	0,63	2,85	2,75	0,10
7	10,71	10,73	0,03	10,95	10,61	0,34
8	10,18	10,00	0,18	10,53	10,04	0,49
9	14,15	14,30	0,15	14,15	14,32	0,17
10	10,00	10,06	0,06	10,00	9,80	0,20
11	14,98	14,82	0,16	14,47	14,34	0,13

4.3 Nadměrné převýšení tupého svaru, 1.9, 502

tento parametr se počítá jako kolmá vzdálenost úsečky mezi dvěma body (levý horní a pravý horní - vzdálenost x_1 nebo levý dolní a pravý dolní - vzdálenost x_2, dle měřeného převýšení) na uživatelem definovaný bod v největší vzdálenosti od těchto bodů, jak je vidno na náčrtu níže27. V níže uvedené tabulce jsou výsledky naměřených hodnot3.



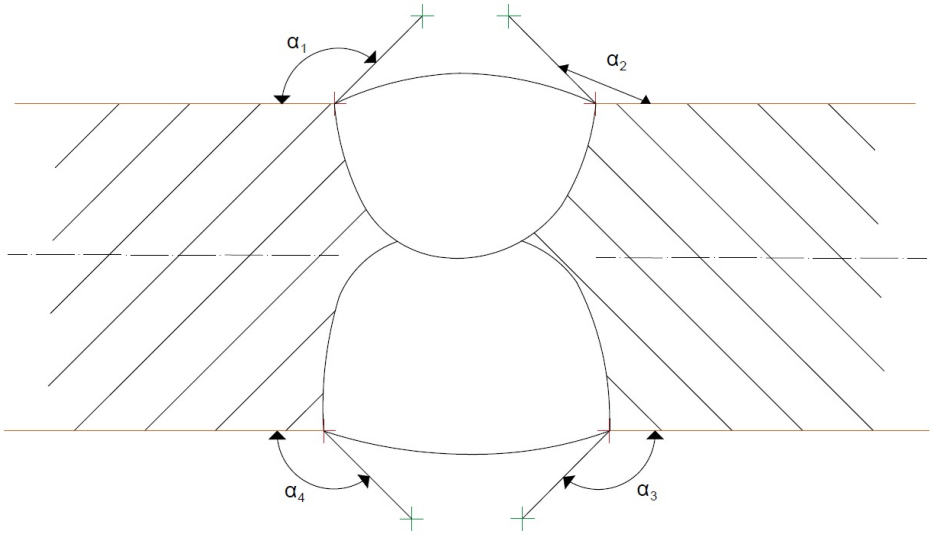
Obrázek 27: Nákres měření nadměrného převýšení tupého svaru

Tabulka 3: Naměřené hodnoty nadměrného převýšení

	Horní			Dolní		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
1	1,90	2,28	0,38	1,20	1,26	0,06
2	0,61	0,77	0,16	1,39	1,23	0,16
3	2,50	2,69	0,19	0,75	0,56	0,19
4	1,18	1,16	0,02	0,53	1,08	0,55
5	0,82	0,66	0,16	0	0	0
6	0,78	0,86	0,08	1,81	1,63	0,18
7	1,67	1,48	0,19	0,71	0,56	0,15
8	1,75	1,48	0,27	1,58	1,48	0,10
9	1,32	1,09	0,23	2,83	2,37	0,46
10	1,82	1,39	0,43	1,27	1,10	0,17
11	1,36	1,36	0	2,72	2,51	0,21

4.4 Strmý přechod svaru, 1.12, 505

Tento parametr se měří jako úhel mezi přímkou aproximující hranu plechu ohraničenou okrajovým bodem svaru a uživatelem definovaným bodem na hraně svaru. Tento úhel se měří pro všechny čtyři okrajové body svaru, naznačeno níže28.



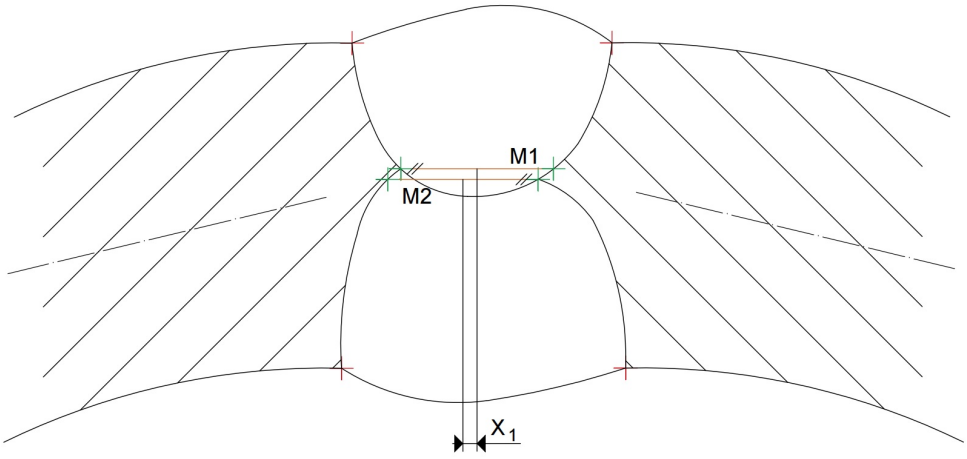
Obrázek 28: Nákres měření strmého přechodu svaru

Tabulka 4: Tabulka naměřených hodnot strmého přechodu svaru (# - číslo vzorku, P - pravítko, A - plikace, O - odchylka)

	Levý horní			Pravý horní			Levý dolní			Pravý dolní		
#	P	A	O	P	A	O	P	A	O	P	A	O
1	135	137	2	118	115	3	141	135	6	159	160	1
2	171	168	3	176	172	4	143	149	6	160	152	8
3	147	149	2	156	157	1	151	153	2	154	154	0
4	172	167	5	59	160	1	176	175	1	172	173	1
5	161	162	1	169	176	7						
6	167	171	4	157	161	4	146	143	3	144	148	4
7	151	156	4	156	159	3	174	176	2	170	168	2
8	140	146	6	154	149	5	149	153	4	157	158	1
9	170	167	3	168	166	2	125	127	2	160	162	2
10	140	138	2	155	157	2	145	142	3	155	155	0
11	163	165	2	131	164	3	126	118	8	163	168	5

4.5 Přesazení

Přesazení se počítá mezi dvěma uživatelem definovanými úsečkami $M1$ a $M2$. Tyto úsečky mají směrnici určenou průměrnou směrnicí materiálů z levé a pravé strany. Přesazení se počítá jako vzdálenost středů přímek $M1$ a $M2$ a to tak, že střed přímky $M1$ je kolmě promítnut na přímku $M2$.



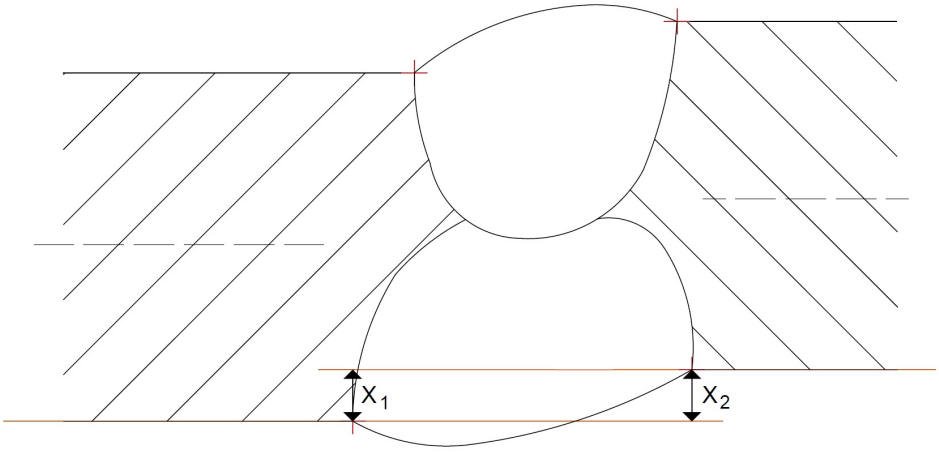
Obrázek 29: Nákres měření přesazení výrobku

Tabulka 5: Tabulka naměřených hodnot přesazení (# - číslo vzorku, P - pravítko, A - plikace, O - odchylka)

	Přesazení		
#	P	A	O
1	0,18	0,09	0,09
2	0,21	0,29	0,08
3	0,35	0,45	0,10
4	0,27	0,36	0,09
5	0,36	0,48	0,12
6	0,39	0,45	0,06
7	2,00	2,50	0,50
8	0,16	0,33	0,17
9	0,50	0,66	0,16
10	0,27	0,36	0,09
11	1,37	1,24	0,13

4.6 Lineární přesazení mezi plechy, 3.1, 5071

Tento parametr se měří na levé a pravé části výrobku. Pro levou část je dán jako kolmá vzdálenost levého dolního bodu svaru a přímky aproximující pravou dolní hranu materiálu. Pravé přesazení se dán jako kolmá vzdálenost pravého dolního bodu svaru a přímky aproximující levou dolní hranu materiálu. Znázorněno na náčrtu níže30.



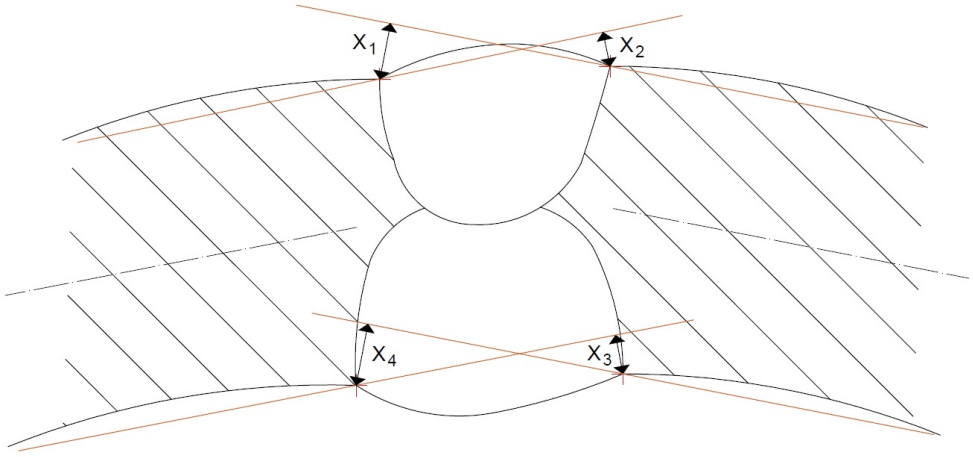
Obrázek 30: Nákres měření lineárního přesazení mezi plechy

Tabulka 6: Tabulka naměřených hodnot lineárního přesazení mezi plechy

	Vlevo			Vpravo		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
1	0,52	0,46	0,06	0,28	0,11	0,17
2	0,14	0,10	0,04	0,56	0,41	0,15
3	0,13	0,28	0,15	0,25	0,31	0,06
4	0	0,02	0,02	0,13	0,20	0,07
5	0,29	0,29	0,02	0,11	0,17	0,06
6	0,26	0,22	0,04	0,39	0,37	0,02
7	0,48	0,67	0,19	1,19	1,16	0,03
8	0,09	0,2	0,11	0,53	0,62	0,09
9	0,38	0,36	0,02	0	0,01	0,01
10	0,90	0,85	0,05	1,82	1,67	0,15
11	0,08	0,13	0,05	0,15	0,18	0,03

4.7 Radiální přesazení mezi plechy

Tento parametr se měří pro každý okrajový bod svaru. Je dán jako kolmá vzdálenost okrajového bodu svaru a přímkou aproximující protilehlou hranu materiálu. Pro pravý horní bod to je kolmá vzdálenost k přímce levé horní hrany. Měřené hodnoty z 31 x_1 - levé vnější radiální přesazení, x_2 - pravé vnější radiální přesazení, x_3 - pravé vnitřní radiální přesazení, x_4 - levé vnitřní radiální přesazení.



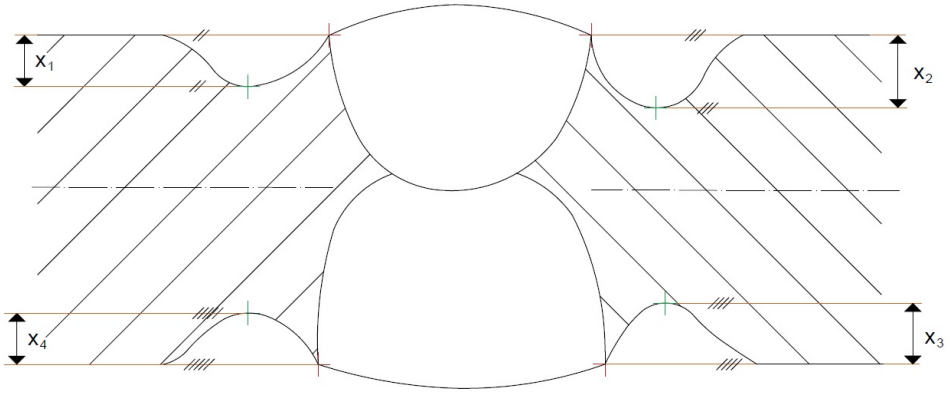
Obrázek 31: Nákres měření radiálního přesazení mezi plechy

Tabulka 7: Tabulka naměřených hodnot radiálního přesazení mezi plechy (# - číslo vzorku, P - pravítko, A - plikace, O - odchylka)

	Levé vnější			Pravé vnější			Levé vnitřní			Pravé dolní		
#	P	A	O	P	A	O	P	A	O	P	A	O
1	1,03	0,56	0,47	1,38	0,84	0,54	0,52	0,46	0,06	0,28	0,11	0,17
2	0,14	0,34	0,20	0,28	0,50	0,22	0	0,10	0,10	0,28	0,41	0,13
3	0	0,06	0,06	0,05	0	0,05	0	0,28	0,28	0,25	0,31	0,06
4	0,14	0,09	0,05	0,07	0,04	0,03	0	0,02	0,02	0,13	0,20	0,07
5	0,47	0,72	0,25	0,23	0,29	0,06	0,23	0,28	0,05	0,23	0,17	0,06
6	0,26	0,19	0,07	0,13	0,10	0,03	0,26	0,23	0,03	0,26	0,37	0,11
7	1,19	1,08	0,11	0,24	0,36	0,12	0,71	0,67	0,04	1,19	1,16	0,03
8	0,18	0,25	0,07	0,35	0,46	0,11	0,18	0,20	0,02	0,53	0,61	0,08
9	0,19	0,21	0,02	0,19	0,26	0,07	0,28	0,36	0,08	0	0	0
10	0,91	0,78	0,13	0,55	0,63	0,08	1,09	0,95	0,14	1,82	1,67	0,15
11	0,34	0,29	0,05	0,09	0,11	0,02	0,17	0,13	0,04	0,17	0,13	0,04

4.8 Souvislý zápal, 1.7, 501

Velikost souvislého zápalu je možno měřit až na čtyřech místech výrobku. Měří se kolmá vzdálenost uživatelem zadaného bodu zápalu a nejbližší přímky hrany výrobku. Dle obrázku32 se měří tyto velikosti zápalu: x_1 - horní levý zápal, x_2 - horní pravý zápal, x_3 - spodní pravý zápal, x_4 - spodní levý zápal



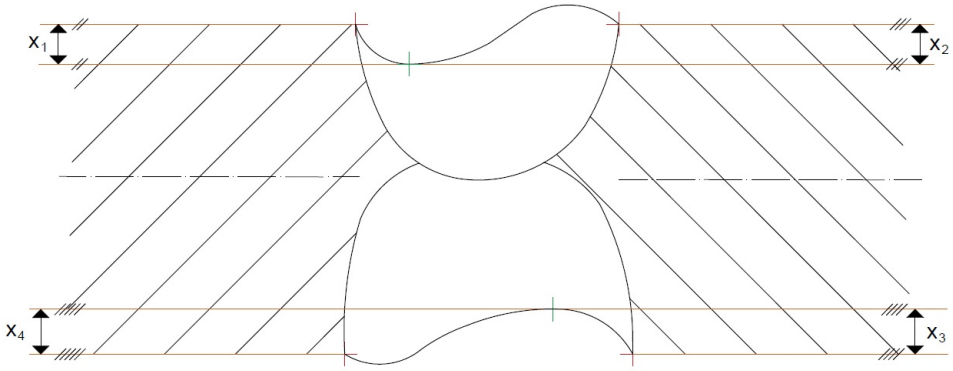
Obrázek 32: Nákres měření souvislého zápalu

Tabulka 8: Tabulka naměřených hodnot souvislého zápalu

	Levý horní			Pravý horní			Levý dolní			Pravý dolní		
#	P	A	O	P	A	O	P	A	O	P	A	O
2				0,69	0,86	0,17						
3				0,58	0,87	0,19						
4										2,27	2,42	0,15

4.9 Proláklina, 1.14, 509

Proláklina se měří na dvou místech x_1 a x_2 a je to kolmá vzdálenost uživatelem definovaného bodu k bližšímu bodu hrany svaru, měřená vůči přímce dvou protilehlých hran materiálu. Tudíž při měření hodní prolákliny se vezme přímka horní levé a horní pravé hrany materiálu a změří se kolmá vzdálenost k bližšímu bodu rohu. Dle obrázku níže33



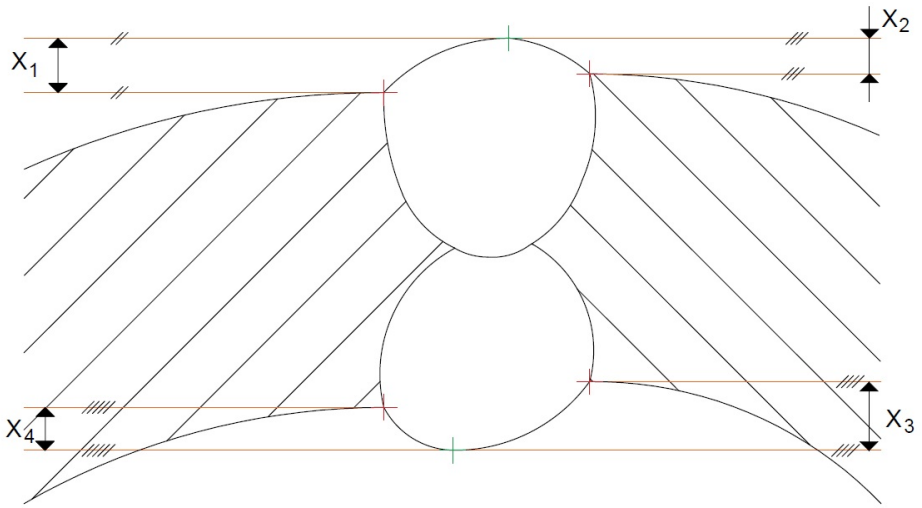
Obrázek 33: Nákres měření prolákliny

Tabulka 9: Tabulka naměřených hodnot pro parametr proláklina

	Horní			Dolní		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
2	0,69	0,65	0,04			
5				0,47	0,21	0,26
7				0,40	0,45	0,05

4.10 Výška svarové housenky

Tento parametr se počítá až na čtyřech bodech a je dán jako kolmá vzdálenost uživatelem definovaného bodu na hraně svaru a přímky reprezentující hranu materiálu. Například výšku vrchní svarové housenky vypočteme jako minimum z dvou kolmých vzdáleností mezi uživatelem definovaného bodu a přímkou aproximující levou horní hranu svařeného materiálu a mezi definovaným bodem a přímkou pro pravou horní hranu materiálu, jak je patrno z následujícího obrázku34



Obrázek 34: Nákres měření výšky svarové housenky

Tabulka 10: Tabulka naměřených hodnot pro výšku svarové housenky

	Horní			Dolní		
Vzorek	Pravítko	Aplikace	Odchylka	Pravítko	Aplikace	Odchylka
1	1,72	1,68	0,04	1,20	1,00	0,20
2	0,56	0,50	0,06	1,39	1,33	0,06
3	2,50	2,44	0,06	0,75	0,71	0,04
4	0,89	1,06	0,17	0,25	0,49	0,24
5	1,17	0,71	0,46	0,12	0,13	0,01
6	0,78	0,83	0,05	2,07	1,92	0,15
7	0,95	0,78	0,17	0,23	0,25	0,02
8	1,23	1,62	0,39	0,70	0,90	0,20
9	0,94	1,05	0,11	1,89	2,10	0,21
10	0,91	0,94	0,03	0,91	0,98	0,07
11	0,85	0,92	0,07	2,21	2,40	0,19

5 Závěr

Výsledkem této práce je aplikace vyhodnocující kvalitativní parametry dle norem ISO 5817 a EN ISO 3183. V začátku textu popisuji metodiky zpracování obrazu. Jednou z popisovaných metod je vyrovnání jasového histogramu, metody na odstranění filtru konvolucí, či bilaterální filtr. Následně jsou rozebrány metody pro určení bodů zájmů ve 2D obraze, přesněji detekce hran, Cannyho hranový detektor. Pak jsou rozebrány některé funkcionality knihovny OpenCV a frameworku Qt, kterých bylo využito v aplikaci. Stručný přehled složení aplikace a její implementace. V poslední části se zaobírá výsledkům měření jednotlivých parametrů jak aplikací, tak ručním způsobem.

Vytvořená aplikace bude mít, na provoze, za následek zefektivnění práce kontroly kvality. Pracovník nebude muset pro každý kontrolovaný výrobek složitě měřit tradičními způsoby a ještě jej pokaždé násobit měřítkem. Zaškolení pracovníka taktéž nebude složité, jelikož je aplikace jednoduchá na ovládání a přehledná. Zaškolenému pracovníkovi bude stačit aplikaci spustit, otevřít příslušný obraz, vybrat měřené parametry a určit jejich body. Následně z tabulky vyčte všechny potřebné parametry.

Literatura

[1] Eduard Sojka *DIGITÁLNÍ ZPRACOVÁNÍ A ANALÝZA OBRAZŮ*, Ostrava, 2000, 133 s., ISBN 80-7078-746-5.

[2] OpenCV *Canny Edge Detector*http://docs.opencv.org/3.1.0/da/d5c/tutorial_canny_detector.html

[3] OpenCV, *http://opencv.org/*

[4] OpenCV *Line Segments Detector*http://docs.opencv.org/3.0-beta/modules/line_descriptor/doc/LSDDetector.html

[5] Qt (knihovna) *Wikipedie. [online]. Dostupné z: https://cs.wikipedia.org/wiki/Qt_(knihovna)*

[6] Signals & Slots *Qt 4.8. Qt Documentation [online]. Copyright © [cit. 22.04.2017]. Dostupné z: http://doc.qt.io/qt-4.8/signalsandslots.html*

[7] All Modules *Qt 5.5 . Qt Documentation [online]. Copyright © [cit. 22.04.2017]. Dostupné z: https://doc.qt.io/archives/qt-5.5/qtmodules.html*

[8] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall *LSD: a Line Segment Detector, Image Processing On Line, 2 (2012), pp. 35–55. https://doi.org/10.5201/ipol.2012.gjmr-lsd*

[9] Ancient Asia *Thresholding: A Pixel-Level Image Processing Methodology Preprocessing Technique for an OCR System for the Brahmi Script*[online]. Copyright ©, [cit. 24.04.2017]. Dostupné z: http://www.ancient-asia-journal.com/articles/10.5334/aa.06113/

[10] OpenCV *Image Thresholding* OpenCV documentation index [online]. Dostupné z: http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html

[11] Nobuyuki Otsu (1979) *A threshold selection method from gray-level histograms* IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66

[12] OpenCV 3.0.0-dev documentation *Morphological Transformations*OpenCV documentation index [online]. Copyright © Copyright 2011 [cit. 25.04.2017]. Dostupné z: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[13] Dr. Leonid P. Yaroslavsky *Digital Picture Processing* (1985), 276 str., ISBN: 0387119345 (U.S.)

[14] Wikipedie *Konvoluce*[online]. Dostupné z: https://cs.wikipedia.org/wiki/Konvoluce

[15] MathWorks *Makers of MATLAB and Simulink*[online]. Copyright © [cit. 26.04.2017]. Dostupné z: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/44648/versions/1/screenshot.jpg

[16] Wikipedia *Histogram equalization* [online]. Dostupné z: https://en.wikipedia.org/wiki/Histogram_equalization

[17] Spatial Filters *Gaussian smoothing*[online]. Copyright ©2003 R. [cit. 26.04.2017]. Dostupné z: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[18] Haishi's Blog *Image processing C# tutorial 4 – Gaussian blur*Haishi's Blog [online]. Dostupné z: http://haishibai.blogspot.cz/2009/09/image-processing-c-tutorial-4-gaussian.html

[19] 88zombie88 *Bilateral Filtering for Gray and Color Image* - CSDN.NET [online]. Dostupné z: http://blog.csdn.net/wangrengxing/article/details/17614293

[20] Sylvain Paris, Pierre Kornprobst, Jack Tumblin and Frédo Durand *Bilateral Filtering: Theory and Applications* Foundations and Trends in Computer Graphics and Vision Vol. 4, No. 1 (2008) 1–73 2009 S. Paris, P. Kornprobst, J. Tumblin and F. Durand DOI: 10.1561/0600000020

[21] PAVEL KUMPÁN *VYTVOŘENÍ 3D MODELU PROSTŘEDÍ POMOCÍ SENZORU KINECT* BAKALÁŘSKÁ PRÁCE, BRNO 2014, str. 45

6 Obsah CD

Na přiloženém CD jsou následující soubory:

1. Zdrojový kód a projekt pro testování
2. Text této diplomové práce ve formátu PDF